Décomposer les monolithes en microservices

AWS Conseils prescriptifs

AWS Conseils prescriptifs: Décomposer les monolithes en microservices

Table of Contents

Décomposer les monolithes en microservices

Tabby Ward et Dmitry Gulin, Amazon Web Services ()AWS

Avril 2023 (historique du document)

La migration vers le cloud Amazon Web Services (AWS) présente de nombreux avantages, notamment l'agilité technique et commerciale, de nouvelles opportunités de revenus et une réduction des coûts. Pour tirer pleinement parti de ces avantages, vous devez continuellement moderniser les logiciels de votre entreprise en refactorisant vos applications monolithiques en microservices. Ce processus comprend trois étapes principales :

- Décomposer les monolithes en microservices : utilisez les modèles de décomposition fournis dans ce guide pour décomposer les applications monolithiques en microservices.
- Intégrer les microservices : intégrez les microservices nouvellement créés dans une architecture de microservices en utilisant AWS des services sans serveur.
- Activez la persistance des données pour les microservices : encouragez la persistance polyglotte entre vos microservices en décentralisant les magasins de données.

La modernisation implique généralement deux types de projets :

- Les projets Brownfield impliquent le développement et le déploiement d'un nouveau système logiciel dans le contexte de systèmes existants ou anciens.
- Les projets Greenfield impliquent la création d'un système à partir de zéro pour un environnement totalement nouveau, sans aucun code existant.

Pour les projets de friche industrielle, l'une des premières étapes de la modernisation de vos applications consiste à décomposer les monolithes de votre portefeuille en microservices.

La plupart des applications commencent par des monolithes conçus pour un cas d'utilisation professionnel spécifique. Si l'architecture du monolithe n'impose pas de conception modulaire, un monolithe peut rester un choix valable pour les applications dont les responsabilités ne sont pas clairement définies dans les limites d'une connaissance bien établie du domaine. La caractéristique centrale d'un monolithe en tant qu'unité de déploiement unique peut également contribuer à atténuer les défauts de conception, tels que le couplage serré ou l'absence de structure interne.

Bien qu'un monolithe puisse être une option valable dans certains cas d'utilisation, il n'est généralement pas adapté à une application moderne. Les structures internes mal définies d'un monolithe peuvent compliquer la maintenance du code, ce qui entraîne une courbe d'apprentissage abrupte pour les nouveaux développeurs et entraîne des coûts de support supplémentaires. Un couplage élevé et une faible cohésion peuvent augmenter considérablement le temps nécessaire pour ajouter de nouvelles fonctionnalités, et il se peut que vous ne puissiez pas adapter les composants individuels en fonction des modèles de trafic. Les monolithes nécessitent également la coordination de plusieurs équipes pour une publication de grande envergure, ce qui augmente le fardeau de la collaboration et du transfert de connaissances. Enfin, vous pouvez constater qu'il devient difficile d'ajouter de nouvelles fonctionnalités ou de créer de nouvelles expériences utilisateur lorsque votre activité ou votre base d'utilisateurs augmente.

Pour éviter cela, vous pouvez utiliser des modèles de décomposition pour décomposer les applications monolithiques, les convertir en plusieurs microservices et les migrer vers une architecture de microservices. Une architecture de microservices structure une application sous la forme d'une série de services faiblement couplés. Les microservices sont conçus pour accélérer le développement de logiciels en permettant des processus de livraison et de déploiement continus (CI/CD).

Avant de commencer le processus de décomposition, vous devez évaluer les monolithes à décomposer. Veillez à inclure des monolithes présentant des problèmes de fiabilité ou de performance, ou à inclure plusieurs composants dans une architecture étroitement couplée. Nous vous recommandons également de bien comprendre le cas d'utilisation commerciale du monolithe, sa technologie et ses interdépendances avec d'autres applications.

Ce guide s'adresse aux propriétaires d'applications, aux chefs d'entreprise, aux architectes, aux responsables techniques et aux chefs de projet. Il décrit les six modèles natifs du cloud suivants utilisés pour décomposer les monolithes, et décrit les avantages et les inconvénients de chacun d'entre eux :

- Décomposer en fonction des capacités de l'entreprise
- Décomposer par sous-domaine
- Décomposer par transactions
- Modèle de service par équipe
- Motif Strangler Fig
- Branche par modèle d'abstraction

Le guide fait partie d'une série de contenus qui couvre l'approche de modernisation des applications recommandée par AWS. La série inclut également :

- Stratégie de modernisation des applications dans le cloud AWS
- Approche progressive de la modernisation des applications dans le cloud AWS
- Évaluation de l'état de préparation à la modernisation des applications dans le AWS cloud
- Intégration de microservices à l'aide de services AWS sans serveur

Résultats commerciaux ciblés

Vous devez vous attendre aux résultats suivants après avoir décomposé vos monolithes en microservices :

- Une transition efficace de votre application monolithique vers une architecture de microservices.
- Ajustements rapides aux fluctuations de la demande commerciale sans interrompre les activités de base, telles que la haute évolutivité, la résilience améliorée, la continuité des livraisons et l'isolation des défaillances.
- Innovation plus rapide, car chaque microservice peut être testé et déployé individuellement.

Modèles pour la décomposition des monolithes

Une fois que vous avez décidé de décomposer un monolithe dans votre portefeuille d'applications, vous devez choisir les modèles de décomposition appropriés et les introduire dans votre organisation.

> **Note**
>
> Vous pouvez utiliser plusieurs modèles pour décomposer un monolithe. Par exemple, vous pouvez décomposer un monolithe par capacité commerciale, puis utiliser le modèle de sous-domaine pour le décomposer davantage.

Rubriques

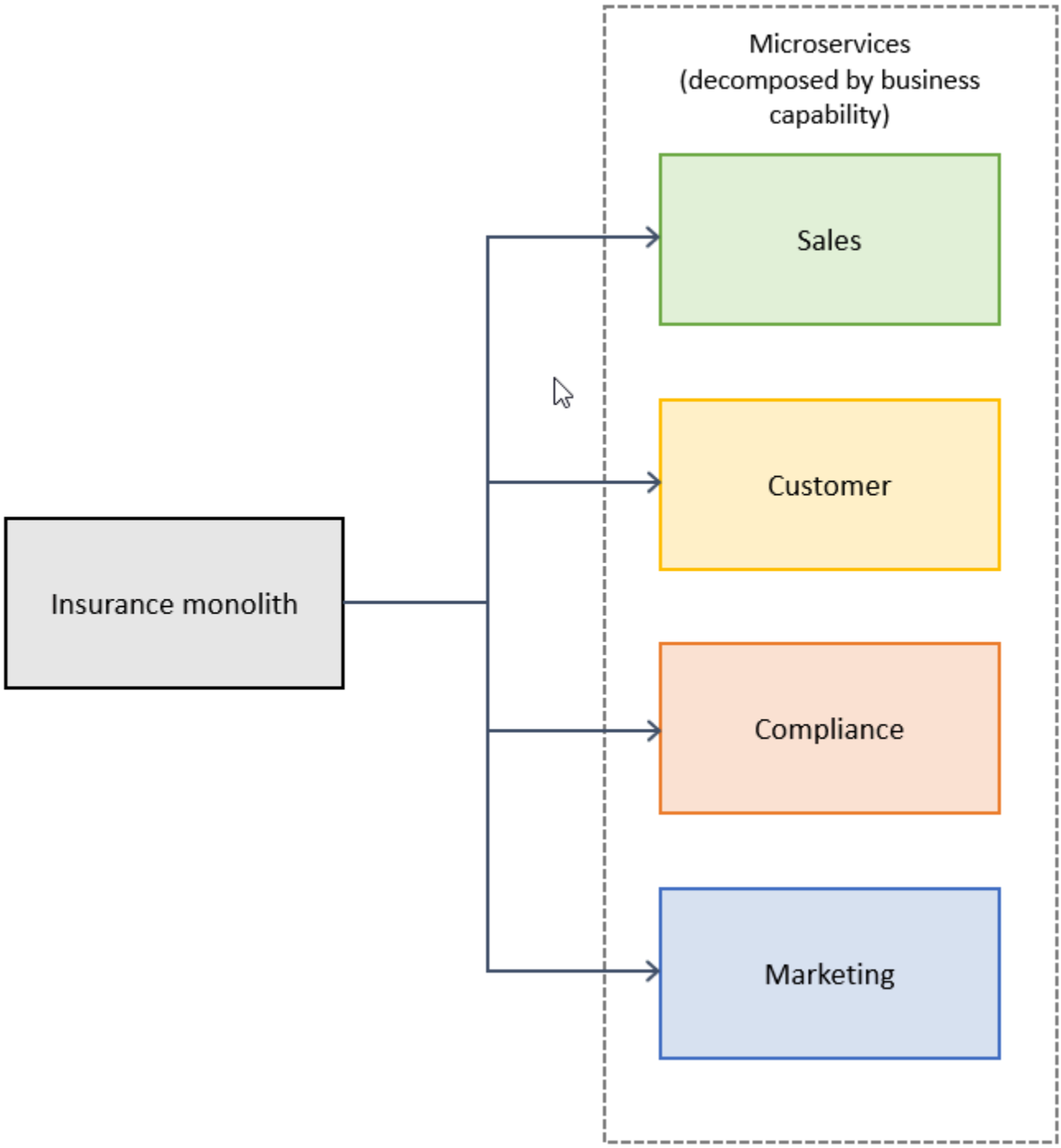
Décomposer selon les capacités de l'entreprise

Vous pouvez utiliser les processus métier ou les capacités de votre organisation pour décomposer un monolithe. Une capacité commerciale est ce qu'une entreprise fait pour générer de la valeur (par exemple, les ventes, le service client ou le marketing). En règle générale, une organisation possède de multiples capacités commerciales, qui varient selon le secteur ou le secteur d'activité. Utilisez ce modèle si votre équipe possède une connaissance suffisante des unités commerciales de votre organisation et si vous disposez d'experts en la matière (PME) pour chaque unité commerciale. Le tableau suivant explique les avantages et les inconvénients de l'utilisation de ce modèle.

Avantages	Inconvénients
• Génère une architecture de microservices stable si les capacités de l'entreprise sont relativement stables.	• La conception des applications est étroiteme nt liée au modèle commercial.

Avantages	Inconvénients
• Les équipes de développement sont interfonctionnelles et organisées de manière à apporter de la valeur commerciale plutôt qu'à des fonctionnalités techniques. • Les services sont faiblement couplés.	• Nécessite une connaissance approfondie de l'ensemble de l'entreprise, car il peut être difficile d'identifier les capacités et les services commerciaux.

Dans le schéma suivant, un monolithe d'assurance est décomposé en quatre microservices basés sur les capacités de l'entreprise.



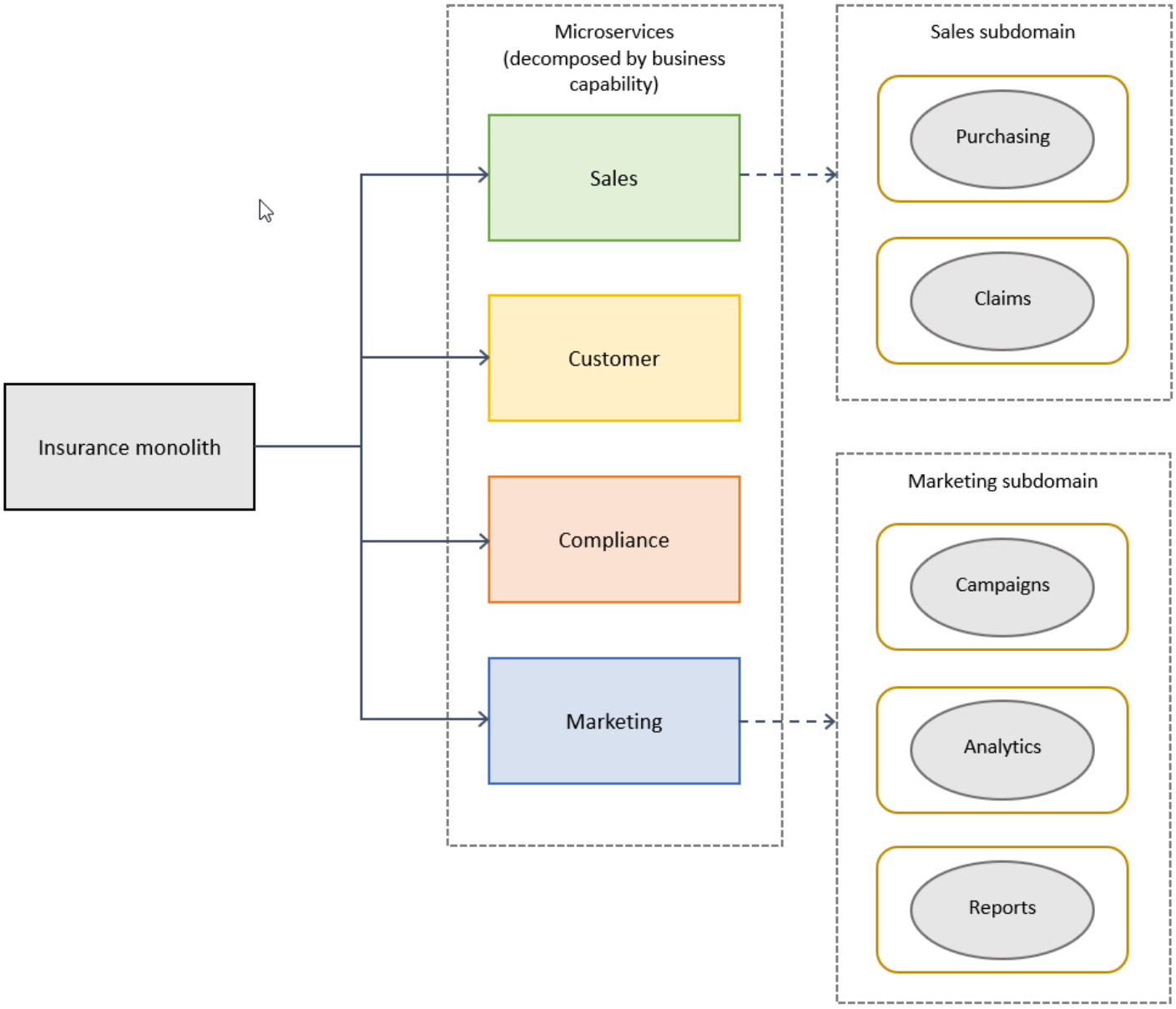
Décomposer par sous-domaine

Ce modèle utilise un sous-domaine de conception pilotée par domaine (DDD) pour décomposer les monolithes. Cette approche décompose le modèle de domaine de l'organisation en sous-domaines distincts qui sont étiquetés comme principaux (un facteur de différenciation clé pour

l'entreprise), support (peut-être lié à l'activité mais pas un facteur de différenciation) ou génériques (non spécifiques à l'entreprise). Ce modèle convient aux systèmes monolithiques existants qui ont des limites bien définies entre les modules liés aux sous-domaines. Cela signifie que vous pouvez décomposer le monolithe en reconditionnant les modules existants sous forme de microservices, mais sans réécrire de manière significative le code existant. Chaque sous-domaine possède un modèle, et la portée de ce modèle est appelée contexte limité. Les microservices sont développés dans ce contexte limité. Le tableau suivant explique les avantages et les inconvénients de l'utilisation de ce modèle.

Avantages	Inconvénients
• L'architecture faiblement couplée assure l'évolutivité, la résilience, la maintenabilité, l'extensibilité, la transparence de l'emplace ment, l'indépendance des protocoles et l'indépendance temporelle. • Les systèmes deviennent plus évolutifs et prévisibles.	• Peut créer trop de microservices, ce qui complique la découverte et l'intégration des services. • Les sous-domaines commerciaux sont difficiles à identifier car ils nécessitent une compréhension approfondie de l'ensemble de l'activité.

L'illustration suivante montre comment un monolithe d'assurance peut être décomposé en sous-domaines après avoir été décomposé par les capacités de l'entreprise.



L'illustration montre que les services de vente et de marketing sont divisés en microservices plus petits. Les modèles d'achat et de réclamation sont des facteurs de différenciation commerciaux importants pour les ventes et sont divisés en deux microservices distincts. Le marketing est décomposé en utilisant des fonctionnalités commerciales complémentaires telles que les campagnes, les analyses et les rapports.

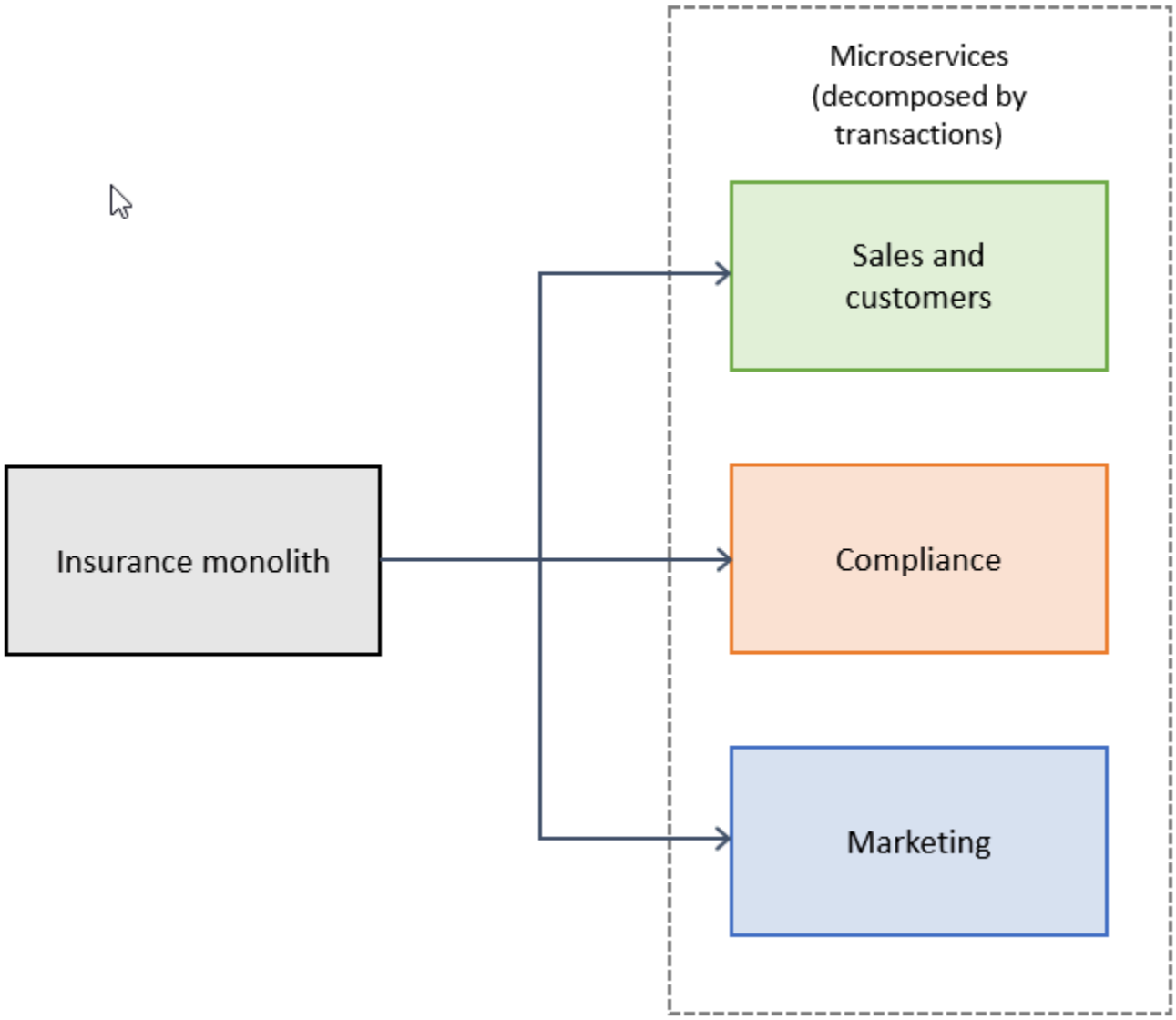
Décomposer par transactions

Dans un système distribué, une application doit généralement appeler plusieurs microservices pour effectuer une transaction commerciale. Pour éviter les problèmes de latence ou les problèmes de

validation en deux phases, vous pouvez regrouper vos microservices en fonction des transactions. Ce modèle est approprié si vous considérez que les temps de réponse sont importants et que vos différents modules ne créent pas de monolithe une fois que vous les avez empaquetés. Le tableau suivant explique les avantages et les inconvénients de l'utilisation de ce modèle.

Avantages	Inconvénients
• Des temps de réponse plus rapides. • Vous n'avez pas à vous soucier de la cohérence des données. • Disponibilité améliorée.	• Plusieurs modules peuvent être regroupés, ce qui peut créer un monolithe. • De multiples fonctionnalités sont mises en œuvre dans un seul microservice au lieu de microservices distincts, ce qui augmente les coûts et la complexité. • Transaction-oriented les microservices peuvent se développer si le nombre de domaines commerciaux et de dépendances entre eux est élevé. • Des versions incohérentes peuvent être déployées en même temps pour le même domaine d'activité.

Dans l'illustration suivante, le monolithe d'assurance est décomposé en plusieurs microservices basés sur les transactions.



Dans un système d'assurance, une demande de réclamation est généralement adressée à un client une fois qu'elle a été soumise. Cela signifie qu'un service de réclamation ne peut exister sans un microservice destiné aux clients. Les ventes et les clients sont regroupés dans un seul ensemble de microservices, et une transaction commerciale nécessite une coordination avec les deux.

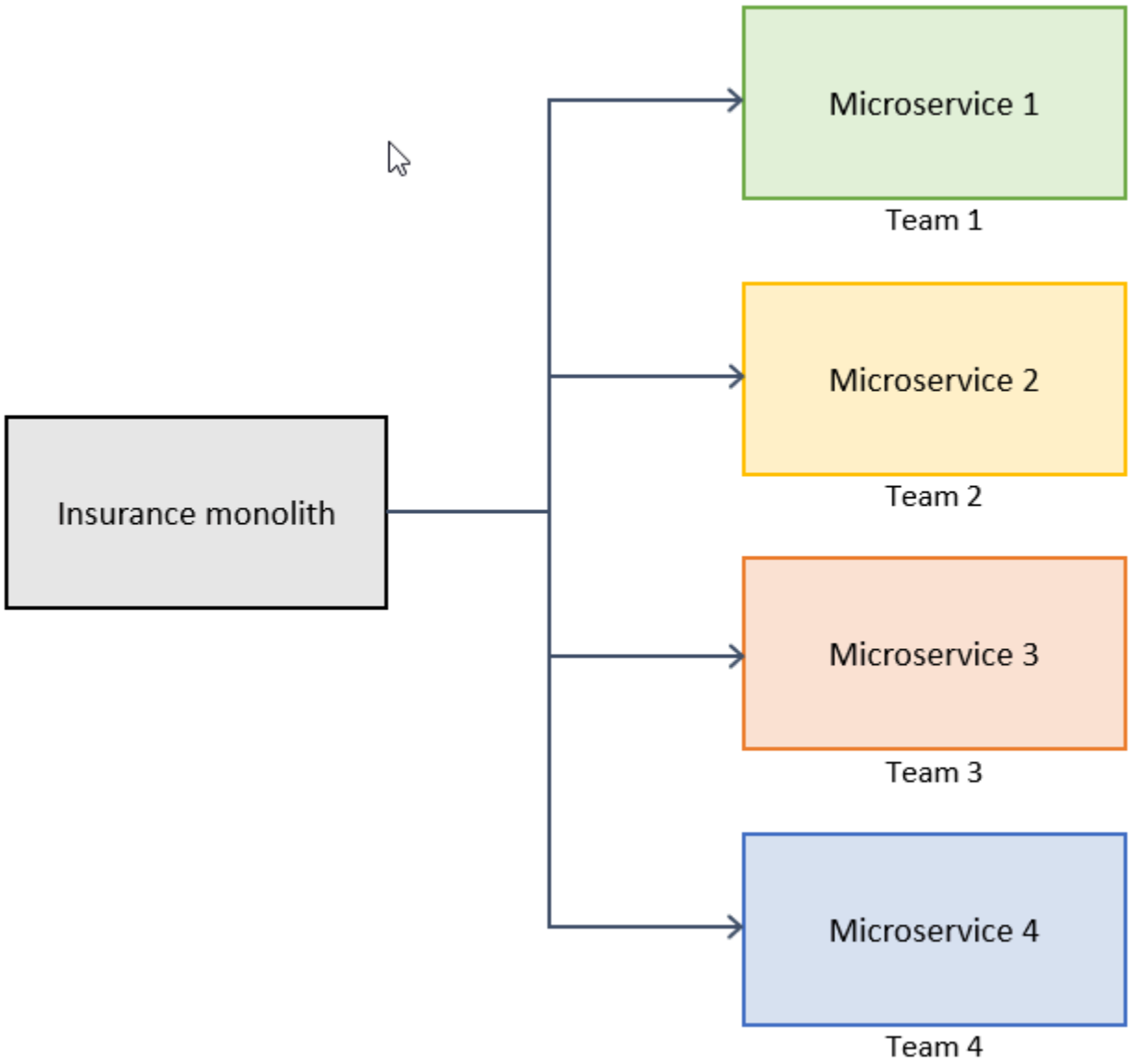
Modèle de service par équipe

Au lieu de décomposer les monolithes en fonction des capacités ou des services de l'entreprise, le modèle de service par équipe les décompose en microservices gérés par des équipes individuelles. Chaque équipe est responsable d'une fonctionnalité commerciale et possède la base de code de cette fonctionnalité. L'équipe développe, teste, déploie ou fait évoluer ses services de manière

indépendante, et interagit principalement avec d'autres équipes pour négocier des API. Nous vous recommandons d'attribuer chaque microservice à une seule équipe. Toutefois, si l'équipe est suffisamment importante, plusieurs sous-équipes peuvent posséder des microservices distincts au sein de la même structure d'équipe. Le tableau suivant explique les avantages et les inconvénients de l'utilisation de ce modèle.

Avantages	Inconvénients
• Les équipes agissent de manière indépendante avec un minimum de coordination. • Les bases de code et les microservices ne sont pas partagés par plusieurs équipes. • Les équipes peuvent rapidement innover et modifier les fonctionnalités des produits. • Les différentes équipes peuvent utiliser des technologies, des frameworks ou des langages de programmation différents. Important : ils doivent être cachés derrière une API publique stable et bien définie.	• Il peut être difficile d'aligner les équipes sur les fonctionnalités de l'utilisateur final ou sur les capacités commerciales. • Des efforts supplémentaires sont nécessaires pour fournir des incréments d'applications plus importants et coordonnés, en particulier s'il existe des dépendances circulaires entre les équipes.

L'illustration suivante montre comment un monolithe peut être divisé en microservices gérés, maintenus et fournis par des équipes individuelles.



Motif Strangler Fig

Les modèles de conception décrits jusqu'à présent dans ce guide s'appliquent aux applications de décomposition pour des projets entièrement nouveaux. Qu'en est-il des projets de friches industrielles impliquant de grandes applications monolithiques ? Il sera difficile de leur appliquer les modèles de conception précédents, car les diviser en petits morceaux lorsqu'ils sont utilisés activement est une tâche ardue.

Le motif Strangler Fig est un motif populaire qui a été introduit par Martin Fowler, qui s'est inspiré d'un certain type de figue qui se répand dans les branches supérieures des arbres. L'arbre existant devient initialement une structure de support pour le nouveau figuier. La figue envoie ensuite ses

racines au sol, enveloppant progressivement l'arbre d'origine et ne laissant à sa place que le nouveau figuier autoportant.

Ce modèle est couramment utilisé pour transformer progressivement une application monolithique en microservices en remplaçant une fonctionnalité particulière par un nouveau service. L'objectif est de faire coexister l'ancienne version avec les nouvelles versions modernisées. Le nouveau système est initialement soutenu par le système existant et le complète. Cette prise en charge donne au nouveau système le temps de se développer et éventuellement de remplacer entièrement l'ancien système.

Le processus de transition d'une application monolithique à des microservices en implémentant le modèle Strangler Fig comprend trois étapes : transformer, coexister et éliminer :

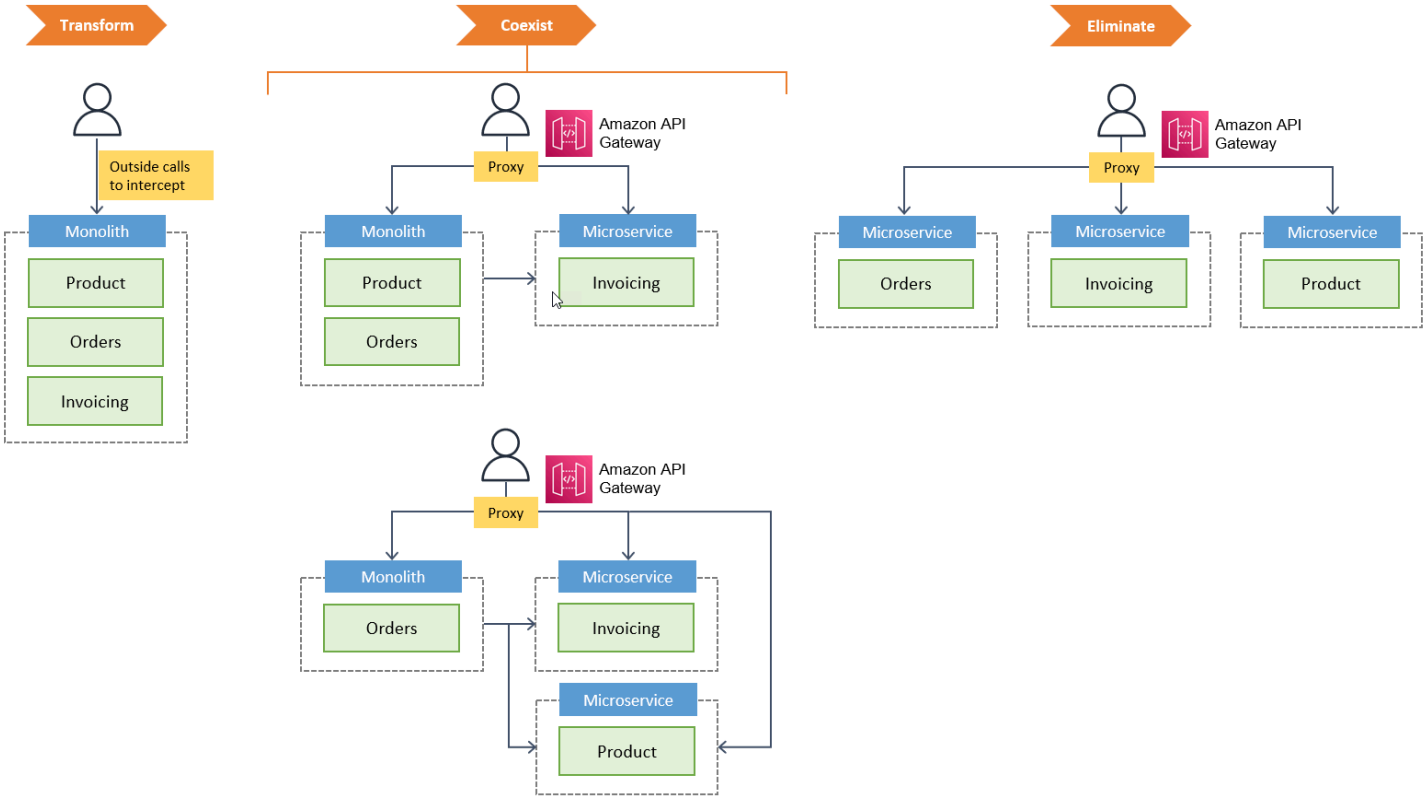
- Transformation : identifiez et créez des composants modernisés en les portant ou en les réécrivant en parallèle avec l'ancienne application.
- Coexister — Conservez l'application monolithe pour la restauration. Interceptez les appels système extérieurs en incorporant un proxy HTTP (par exemple, Amazon API Gateway) au périmètre de votre monolithe et redirigez le trafic vers la version modernisée. Cela vous permet de mettre en œuvre les fonctionnalités de manière incrémentielle.
- Élimination : retirez l'ancienne fonctionnalité du monolithe à mesure que le trafic est redirigé du monolithe existant vers le service modernisé.

Le tableau suivant explique les avantages et les inconvénients de l'utilisation du motif Strangler Fig.

Avantages	Inconvénients
• Permet une migration progressive d'un service vers un ou plusieurs services de remplacement. • Permet de conserver les anciens services tout en les refactorisant vers des versions mises à jour. • Permet d'ajouter de nouveaux services et fonctionnalités tout en refactorisant les anciens services. • Le modèle peut être utilisé pour le versionne ment des API.	• Ne convient pas aux petits systèmes peu complexes et de petite taille. • Ne peut pas être utilisé dans les systèmes où les demandes adressées au système principal ne peuvent pas être interceptées et acheminées. • La couche proxy ou de façade peut devenir un point de défaillance unique ou un obstacle aux performances si elle n'est pas conçue correctement. • Nécessite un plan de réduction pour chaque service refactorisé afin de revenir à l'ancienn

Avantages	Inconvénients
• Le modèle peut être utilisé pour les interacti ons existantes avec des solutions qui ne sont pas ou ne seront pas mises à niveau.	e méthode de travail rapide et sûre en cas de problème.

L'illustration suivante montre comment un monolithe peut être divisé en microservices en appliquant le modèle Strangler Fig à une architecture d'application. Les deux systèmes fonctionnent en parallèle, mais vous allez commencer à déplacer les fonctionnalités en dehors de la base de code monolithe et à les améliorer grâce à de nouvelles fonctionnalités. Ces nouvelles fonctionnalités vous permettent de concevoir des microservices de la manière la mieux adaptée à vos besoins. Vous allez continuer à supprimer les fonctionnalités du monolithe jusqu'à ce qu'il soit entièrement remplacé par des microservices. À ce stade, vous pouvez éliminer l'application monolithe. Le point essentiel à noter ici est que le monolithe et les microservices vont cohabiter pendant un certain temps.



Branche par modèle d'abstraction

Le motif Strangler Fig fonctionne bien lorsque vous pouvez intercepter les appels au périmètre du monolithe. Toutefois, si vous souhaitez moderniser des composants qui existent plus profondément

dans la pile d'applications existante et qui ont des dépendances en amont, nous vous recommandons d'utiliser le modèle de branche par abstraction. Ce modèle vous permet d'apporter des modifications à la base de code existante pour permettre à la version modernisée de coexister en toute sécurité avec l'ancienne version sans provoquer de perturbations.

Pour utiliser correctement le modèle de branche par abstraction, procédez comme suit :

1. Identifiez les composants monolithes qui ont des dépendances en amont.
2. Créez une couche d'abstraction qui représente les interactions entre le code à moderniser et ses clients.
3. Lorsque l'abstraction est en place, modifiez les clients existants pour qu'ils utilisent la nouvelle abstraction.
4. Créez une nouvelle implémentation de l'abstraction avec les fonctionnalités retravaillées en dehors du monolithe.
5. Passez l'abstraction à la nouvelle implémentation lorsque vous êtes prêt.
6. Lorsque la nouvelle implémentation fournit toutes les fonctionnalités nécessaires aux utilisateurs et que le monolithe n'est plus utilisé, nettoyez l'ancienne implémentation.

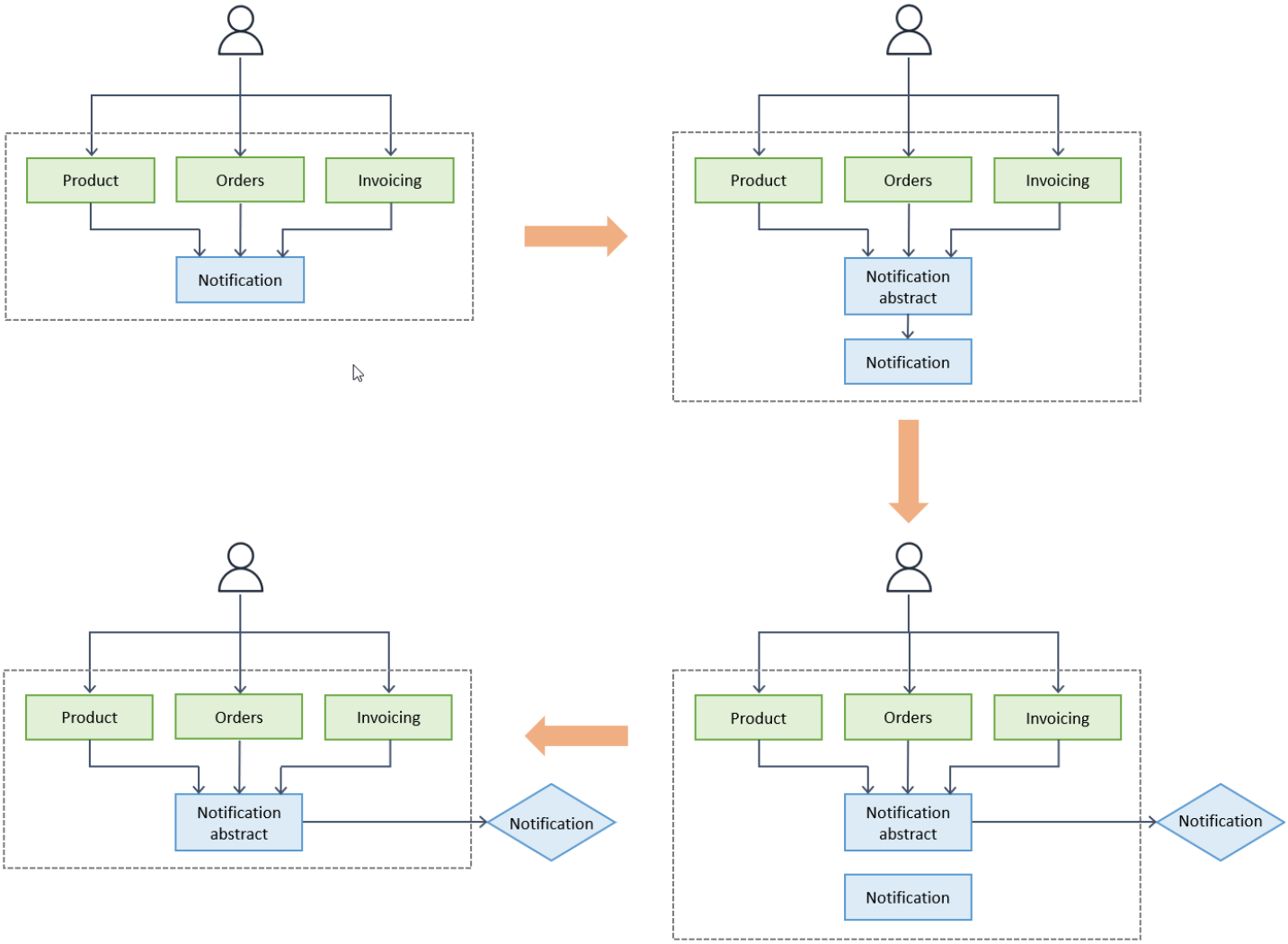
Le modèle de branche par abstraction est souvent confondu avec les bascules de fonctionnalités, qui vous permettent également d'apporter des modifications à votre système de manière incrémentielle. La différence est que les boutons de fonctionnalité sont destinés à permettre le développement de nouvelles fonctionnalités et à les rendre invisibles pour les utilisateurs lorsque le système est en cours d'exécution. Les boutons de fonctionnalité sont donc utilisés au moment du déploiement ou de l'exécution pour choisir si une fonctionnalité ou un ensemble de fonctionnalités en particulier est visible dans l'application. Le branchement par abstraction est une technique de développement qui peut être combinée à des bascules de fonctionnalités pour passer de l'ancienne à la nouvelle implémentation.

Le tableau suivant explique les avantages et les inconvénients de l'utilisation de la branche par modèle d'abstraction.

Avantages	Inconvénients
• Permet des modifications incrémentielles qui sont réversibles en cas de problème (rétrocompatible).	• Ne convient pas si la cohérence des données est impliquée.

Avantages	Inconvénients
• Permet d'extraire les fonctionnalités qui se trouvent au plus profond du monolithe lorsque vous ne pouvez pas intercepter les appels qui lui sont adressés au bord du monolithe. • Permet à plusieurs implémentations de coexister dans le système logiciel. • Fournit un moyen simple de mettre en œuvre un mécanisme de secours en utilisant une étape de vérification intermédiaire pour appeler à la fois les nouvelles et les anciennes fonctionnalités. • Soutient la livraison continue, car votre code fonctionne à tout moment tout au long de la phase de restructuration.	• Nécessite des modifications du système existant. • Cela peut alourdir le processus de développement, en particulier si la base de code est mal structurée. (Dans de nombreux cas, les avantages en valent la peine, et plus la restructuration est importante, plus il est important d'envisager d'utiliser la branche par modèle d'abstraction.)

L'illustration suivante montre le modèle de branche par abstraction pour un composant de notification dans le monolithe d'assurance. Cela commence par créer un résumé ou une interface pour la fonctionnalité de notification. Par petits incréments, les clients existants sont modifiés pour utiliser la nouvelle abstraction. Cela peut nécessiter de rechercher dans la base de code les appels aux API liés au composant Notification. Vous créez la nouvelle implémentation de la fonctionnalité de notification sous forme de microservice en dehors de votre monolithe et vous l'hébergez dans l'architecture modernisée. À l'intérieur de votre monolithe, votre interface d'abstraction nouvellement créée agit comme un courtier et annonce la nouvelle implémentation. Par petites étapes, vous transférez la fonctionnalité de notification vers la nouvelle implémentation, qui reste inactive jusqu'à ce qu'elle soit complètement testée et prête. Lorsque la nouvelle implémentation est prête, vous changez d'abstraction pour l'utiliser. Vous devriez utiliser un mécanisme de commutation qui peut être facilement inversé (comme une bascule de fonctionnalité) afin de pouvoir revenir facilement à l'ancienne fonctionnalité en cas de problème. Lorsque la nouvelle implémentation commence à fournir toutes les fonctionnalités de notification à vos utilisateurs et que le monolithe n'est plus utilisé, vous pouvez nettoyer l'ancienne implémentation et supprimer tout indicateur de fonctionnalité de commutation que vous auriez pu implémenter


Product
Orders
Invoicing
Notification
Product
Orders
Invoicing
Notification abstract
Notification
Product
Orders
Invoicing
Notification abstract
Notification
Product
Orders
Invoicing
Notification abstract
Notification
Notification

FAQ sur la décomposition des monolithes

Cette section fournit des réponses aux questions fréquemment posées sur la décomposition des monolithes.

Pouvez-vous utiliser plusieurs modèles pour décomposer un monolithe ?

Oui, vous pouvez utiliser plusieurs modèles pour décomposer un monolithe. La méthode la plus courante consiste à décomposer un monolithe selon le modèle de décomposition par capacité métier, puis à utiliser le modèle de décomposition par sous-domaine pour le décomposer davantage.

Comment la décomposition d'un monolithe en microservices affecte-t-elle le processus ? DevOps

Comme il n'est pas nécessaire de tout redéployer une fois qu'une modification a été apportée à l'application, vous devez bénéficier du support et de la propriété des microservices nouvellement créés qui sont ajoutés au processus de déploiement. Cela pourrait rendre le DevOps processus plus complexe.

Ressources

Guides associés

- Stratégie de modernisation des applications dans le cloud AWS
- Approche progressive de la modernisation des applications dans le cloud AWS
- Évaluation de l'état de préparation à la modernisation des applications dans le AWS cloud
- Intégration de microservices à l'aide de services AWS sans serveur

Autres ressources

- Divisez une application monolithique en microservices avec AWS Copilot, Amazon ECS, Docker et AWS Fargate

Historique du document

Le tableau suivant décrit les modifications importantes apportées à ce guide. Pour être averti des mises à jour à venir, abonnez-vous à un fil RSS.

Modification	Description	Date
Ajout de nouveaux motifs	Ajout d'informations sur la figure et la branche de l'étrangleur par des motifs d'abstraction.	6 avril 2023
Publication initiale	—	11 janvier 2021

Les traductions sont fournies par des outils de traduction automatique. En cas de conflit entre le contenu d'une traduction et celui de la version originale en anglais, la version anglaise prévaudra.